DDoS detection based on traffic profiles

by

Morten Kråkvik

Agder University College
Faculty of Engineering and Science
Grimstad, Norway
May 2006

Master's Thesis in Information and Communication Technology

Abstract

Distributed denial of service attacks has become a significant threat against Internet resources. These attacks aims at disrupting the victim's service by commanding a massive number of compromised sources to send useless data towards the victim. The distributed nature of these attacks usually makes mitigation a time consuming process, and the risk of collateral damage is high.

In this thesis I propose a method for detecting and identifying the sources of DDoS attacks based on research in the field of network traffic measurement and source IP address monitoring. The method consists of two parts; a network traffic collector and a traffic profile analyser, where the first part is responsible for creating traffic profiles representing the network pattern over certain time periods, and the second part responsible for the analysis.

A novelty in this thesis is the usage of learning automata for tracking the behaviour of source- IP addresses and subnets.

I have shown that when using a specific reinforcement algorithm for the learning automata, the proposed method is able to correctly identify- and distinguish sources participating in distributed denial of service attacks and sources generating normal traffic. It has also been shown that this algorithm is robust against attacks based on IP spoofing.

Due to the fact that the method is tracking both source IP addresses as well as their subnets, more efficient filtering rules can be created based on subnets instead of multiple IP addresses.

Preface

This thesis is submitted to Agder University College, Faculty of Engineering and Science, in partial fulfilment of the degree of Master of Science in Information and Communication Technology.

This work has been carried out in collaboration with Telenor Security Services in Arendal, under the supervision of Nils Ulltveit-Moe at Agder University College and Per Kristian Johnsen at Telenor Security Services.

I would like to thank my supervisors, Nils Ulltveit-Moe and Per Kristian Johnsen for their valuable suggestions and constant help during the research. I also want to thank associate professor Ole-Christoffer Granmo at Agder University College for helpful discussions and advices on learning automata.

Grimstad, May 2006.

Morten Kråkvik

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Distributed Denial of Service attacks, also known as *DDoS* attacks, has become a threat against anyone on the Internet.

Especially, these attacks are an increasing threat against companies and organisations who expose services for their customers, and who rely on the Internet to conduct their everyday businesses.

Most attackers involved in computer criminality seek to break into systems in an attempt to install their software and steal secret user information, such as your passwords and credit card numbers. The goal and effect of a denial of service attack, however, differs from these kinds of attacks. No data is stolen and no software will be installed on the victim host. The one and only goal is to take down the victim's service.

So, how can an attacker take down your service? As illustrated in figure 1.1 on page 3, the attacker has typically broken into several thousand personal computers, located all around the world, and turned them into *bot agents*, or *zombies*. By doing this, the attacker creates a *botnet* allowing him, or her, to gain full control over the compromised computers from a central location known as the *botnet server*. By connecting to this server, the attacker then issues commands to all bot agents at once, telling them to attack a specific address on the Internet. What follows is a storm of useless data from all bot agents headed against the victim, causing legitimate data to the victim to be dropped.

Individually, each compromised host is usually not able to successfully attack a victim due to limited bandwidth. However, when there are several thousands of bot agents, *together* they aggregate massive bandwidth, being able to knock most networks off-line.

The complex nature of these attacks usually makes mitigation a difficult- and time consuming process, and might cause the victims both financial and reputational damages.

Telenor[15], being one of the largest ISP's in Scandinavia, wish to look at possible solutions for quickly detecting DDoS attacks, and identifying the sources participating in the attacks.

The field of DDoS contains many attack- and defence methods, and [8] attempts to give a structured overview of the knowledge in this field. Of

particular interest, considering this thesis, are the sections on response strategies for victim hosts *reacting* to attacks, as opposed to *preventing* attacks. The difference in *reacting to* versus *preventing* DDoS attacks, is that the former method aims at alleviating the impact of an attack while the latter method aims at eliminating the possibility of an attack. Although this thesis will not focus on how to actually mitigate attacks, but rather on attack detection and identification of sources, it will be kept in mind during the work that a reactive method like applying IP address filtering on routers will be used in the case of an attack.

Methods for defending against attacks, like [1, 4, 5], suggests replacing network devices, such as routers, with new «intelligent» hardware which does real-time traffic profile analysis and mitigation in addition to their intended task. A downside of these methods are that in case of an attack, or just under heavy load, such devices could use more resources on analysing the traffic rather than doing it's intended task, which might affect the overall network performance. Also, in cases where the network is very large and complex, replacing network hardware might not be a financially realistic option. In this thesis, the proposed solution will consist of a *passive* sensor located near the victim network. By passive, it means that the device will not affect the network traffic directly, it will only receive a copy of the traffic and not be able to block anything itself.

[8] divides detection methods into two sections; *anomaly-* and *pattern-*based detection. Anomaly-based detection creates profiles of the network traffic and uses this as a base to spot anomalies and identify potential attacks. Proposed solutions taking advantage of this method includes [1, 5]. Pattern-based detection, on the other hand, uses predefined signatures to recognise attack traffic, typically by looking at the packet payload. An example of a system using pattern-based detection is the network intrusion detection *Snort*[12]. In this thesis, the detection method will be based on anomaly detection.

A field which is closely related to DDoS detection is network traffic accounting. Traffic accounting has primarily been used for usage-based network billing and bandwidth provisioning. Applications performing traffic accounting, which usually are the routers, sees the traffic as a collection of flows which needs to be measured. However, as the traffic rate and the number of flows increases (which is the case under an attack), keeping track of the flows becomes very expensive in terms of memory and CPU.

In [3], Estan et al. claims that the currently state-of-the-art accounting methods, Cisco IOS's *NetFlow*[2], which count periodically sampled packets are slow, inaccurate and resource intensive. A novel sampling method has been proposed in [3] which aims at identifying large flows and ignoring the small. This sampling method has been adopted in this thesis and will be described in detail in section 2.2.1 on page 7.

A data-structure called *MULTOPS* has been proposed in [5] by Gil et al. This data-structure was primarily designed for network devices for detecting DDoS attacks by monitoring incoming- and outgoing traffic rates. If a dis-proportionality occurred between receive- and transmit-rate, the event

was flagged as anomalous. The core of this data-structure has been adopted and expanded in this thesis due to its performance and flexibility. The MULTOPS data-structure and the extension will be described in detail in sections 2.2.2 on page 9 and 3.5 on page 16, respectively.

A novelty in this thesis is the usage of *learning automata*[9] for source identification. Learning automata are small decision making devices which are able to operate under non-deterministic environments and improve their performance using past experience. Based on whether a source address appears to be a part of a possible attack, or not, the automaton for the given source address can be rewarded or penalised according to it's updating algorithm. By using this approach, I aim to identify source addresses and subnets participating in an attack, and which are or not. This method will be further discussed in detail in section 4.3 on page 23.

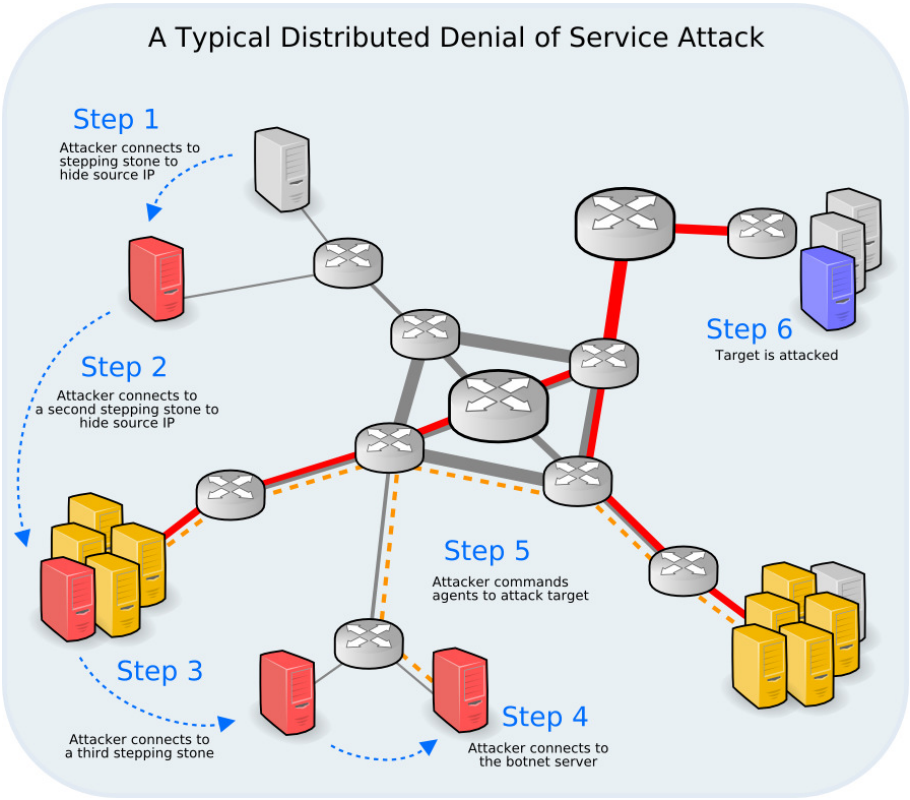

Figure 1.1: An illustration of a typical DDoS attack

1.1 Thesis definition

The final thesis definition was formulated like this:

> «*In this thesis, we will approach the problem of detecting DDoS attacks by constructing traffic profiles representing the traffic pat-*

tern over time. By evaluating these traffic profiles, we aim to look for anomalies and identify potential attacks.»

This definition has been broken into two parts, where both will be addressed in this thesis:

1. **Network traffic accounting**
   In order to detect an attack, network data needs to be captured. The capturing, or the accounting, needs to perform real-time monitoring of the network device and store this data in a sensible format for later analysis. This format, will later be referred to as the *traffic profile*. The traffic profile and the network traffic accounter will be developed as a part of the thesis work. Previous research in the area of traffic measurement and accounting will be outlined in chapter 2 on page 6.

2. **Traffic profile analyser**
   A traffic profile analyser, *TPA*, will be developed as a part of the thesis work. The purpose of the TPA is to continuously analyse the profiles as they are created by the network traffic accounter. These profiles will serve as a base to determine if a network is under attack or not, and to determine who is participating in the attack and who is not. Previous research in the area of DDoS detection will be outlines in chapter 2 on page 6. My contribution in this field of research will be exploring the usage of learning automata for DDoS attack source identification.

1.2 Delimitations

Due to limited resources, this thesis will only focus on detecting distributed denial of service attacks against victim networks. This means that SYN flood attacks, which are typically aimed at exhausting the resources of a specific victim server, and not the network resources itself, are not covered in specific in this thesis. However, if a SYN flood attack allocates enough bandwidth to threaten the network infrastructure, the attack is considered an attack against the network.

This thesis also focuses on detecting attacks near the victim, as opposed to near source end.

1.3 Report outline

Chapter 1 is the introduction chapter, which you are currently reading.

Chapter 2 covers the related work this thesis has been based on. This includes an overview of DDoS attacks and defence methods, research on traffic measurement and DDoS traffic characteristics.

Chapter 3 is dedicated to the development of the network traffic collector. This includes requirements and delimitations of the collector followed by

the choice of sampling algorithm. Next, the extension of the MULTOPS data-structure is described and how this is used to create a traffic profile.

Chapter 4 is dedicated to the development of the traffic profile analyser. This chapter discusses the choice of attributes that are interesting considering indications of a DDoS attack. This chapter also covers the use of learning automata for tracking source address behaviour.

Chapter 5 describes the experiment setup and simulations of the network traffic accounter and traffic profile analyser.

Chapter 6 presents the results of the simulations.

Chapter 7 discusses the results achieved in this thesis.

Chapter 8 is the final chapter which concludes the thesis in addition to looking at possible further work.

Chapter 2

Related work

This chapter is a short introduction to DDoS attacks covering the mainly used attacker techniques in addition to relevant defence methods.

2.1 An overview of denial of service attack methods

2.1.1 Source address spoofing

Source address spoofing, also known as IP spoofing, is a technique which is commonly used by attackers. This technique can, in some cases, be used to gain unauthorised access to a computer by sending messages with an IP address indicating that it is coming from a trusted host. This is accomplished by modifying the source IP address field in the packet's IP header. However, this technique can also be very useful when engaging a denial of service attack. By using this technique you can make a denial of service attack with very few hosts, look distributed to the victim. On a single host, with one IP address, you can replace the source IP address of any packet with a random address. For an attacker, this has the following advantages:

- at the victim, it looks like the attack is distributed, which makes it very difficult to filter based on IP address.
- the actual IP address of the attacking host is not revealed, causing preventive methods like shutting down the attacking host very difficult.

Another advantage (for the attacker) for using IP spoofing is that it can be used to exploit known weaknesses of IP protocols, such as TCP and UDP. This technique is called reflection attack, and will be described in the following section.

2.1.2 DDoS attacks

In this section I will describe some of the mainly used DDoS attack methods.

SYN-flooding

SYN-flood attacks are based on exploiting a well known weakness in the implementation of the TCP protocol. By starting the first phase of the three-way handshake, which is done by sending a single SYN-packet against the server, a resource allocation is made on the server. If this handshake is never completed, these resources are not freed until a timeout has been reached. If an attacker rapidly sends multiple SYN-packets against the victim, this might result in a resource exhaustion at the victim, causing a denial of service effect. This technique is usually combined with IP spoofing, which makes sure the handshake is never completed as the spoofed source will simply discard an out-of-state packet from the victim server.

UDP-flooding

UDP-flooding attacks are classified as bandwidth attacks. These types of attacks aims at simply flooding the victim's network pipe, causing packets to be dropped.

Reflection and amplification attacks

Reflection attack is a result of IP spoofing and is probably the most popular attack method today. This attack aims at flooding the victim with useless packets by sending request packets to innocent third-party servers using the victim's source address. These servers faithfully responds to these packets by sending them back to the spoofed victim. If there are many such servers, with well provisioned links, this will effectively cause a denial of service at the victim.

Especially, DNS-servers are very popular considering reflection attacks. By sending a valid DNS request with spoofed source IP address, the response will be sent to the victim host. If this DNS server is (mis-)configured to accept recursive requests, it can be tricked to cache a very large DNS record, which in turn will be sent to the victim. This is known as a DNS amplification attack and is illustrated in figure 2.1 on the next page. Unfortunately, there are many DNS servers which accept recursive requests, and they are usually well provisioned as well. And since this is all legal traffic, blocking these requests and responses without inflicting collateral damage is very difficult. A study of these attacks has been carried out in [17].

2.2 Traffic measurement and accounting

2.2.1 Sample and hold

As described in [3], the easiest way to identify large flows is through sampling, however with a twist. As with common sampling methods such as [2], each packet is sampled with a probability. If a packet is sampled and the flow it belongs to is not in the memory, a new flow-entry is created.

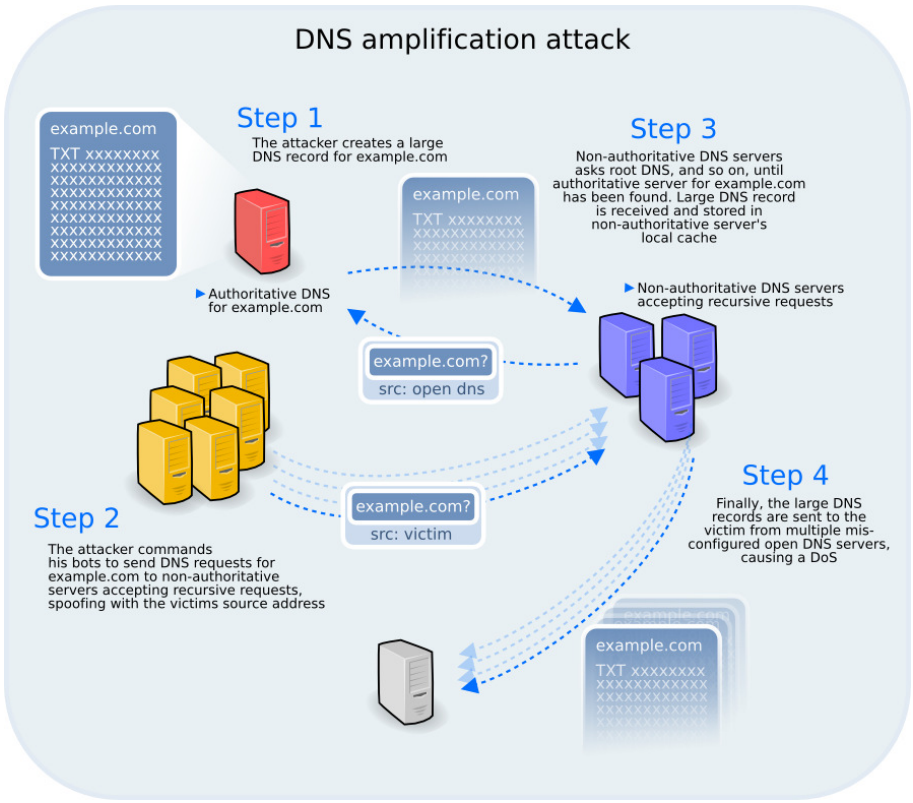

Figure 2.1: DNS amplification attack

When a flow-entry has been created, unlike [2], the *sample-and-hold* algorithm will update the flow statistics for every packet in this flow. Another difference from [2] is that the probability of sampling a packet is a function of the packet size. The sampling probability for packet with size s is $p_s = 1 - (1 - p)^s \approx 1 - e^{-sp}$ which can be approximated by $p_s = p * s$. This sampling algorithm can be described with the pseudo code shown in algorithm 1.

Algorithm 1 The sample-and-hold algorithm

Require: Probability p for sampling a packet
Require: Packet size s
  for each arriving packet **do**
    if packet has a flow-entry **then**
      increment flow byte count with packet size
    else
      add new flow-entry with probability $p_s = p * s$
    end if
  end for

2.2.2 MULTOPS

In [5] Thomer M. Gil et al. proposes a heuristic and a data-structure for network devices, such as routers and network monitors, to detect and mitigate DDoS attacks. Their method proposes to maintain a data-structure called MULTOPS on each network device. Using this data-structure, they attempt to monitor incoming and outgoing packet rates. As shown in figure 2.2 on the next page, MULTOPS (MUlti Level Tree for Online Packet Statistics) is basically a tree of tables and records, where each record contains packet rate statistics for a subnet prefix at different aggregation levels. The data-structure will always have a root `Table`, containing references to 256 subnet `Records` (i.e `[0-255.*.*.*]`). Now, if the packet rate statistics for a subnet exceeds a given threshold value, the tree will expand to include more detailed statistics for this subnet (i.e granularity will be increased). And if the packet rate statistics should go below a given threshold, the tree will contract. This behaviour can be seen as «zooming in and out», respectively.

The heuristic in this method is based on that under normal traffic, the packet flow in one direction is proportional to the flow in the opposite direction.

This method uses disproportional packet rates to or from hosts and subnets to detect and stop attacks. This packet rate statistic can be used to (1) identify the victim of a DDoS attack or (2) identify the sources of an attack. These modes are called *victim-oriented* mode and *attacker-oriented* mode, respectively.

The difference between these modes are important when it comes to blocking traffic. In victim-oriented mode, attack mitigation is done by dropping packets against the victim. In attacker-oriented mode, the mitigation

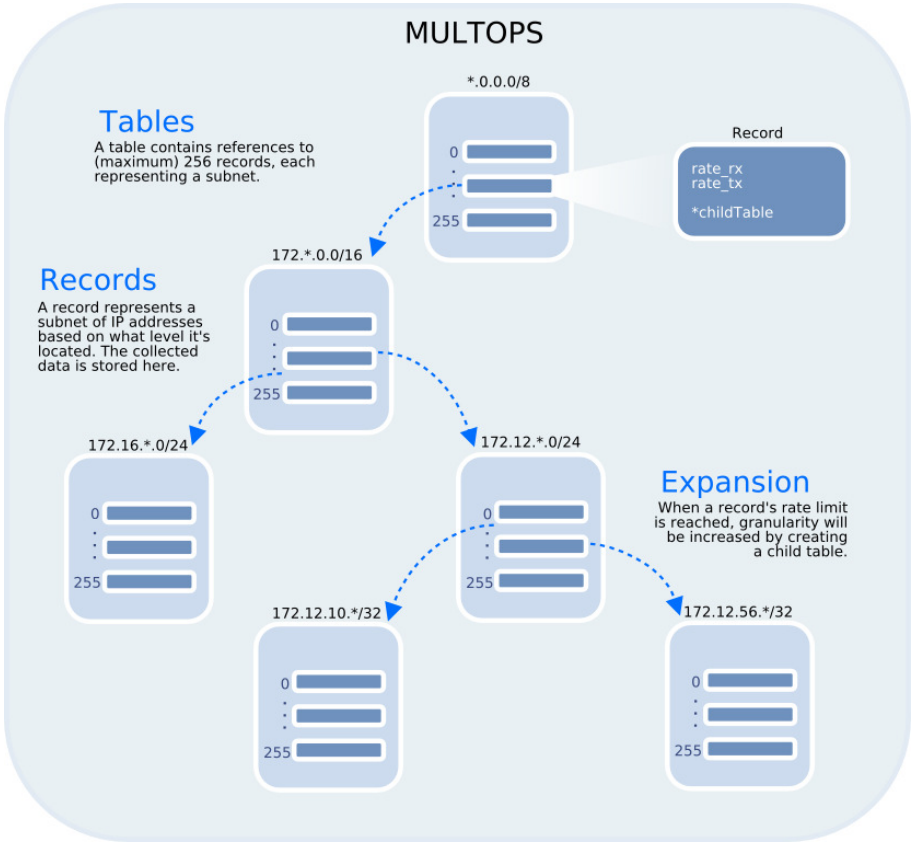

Figure 2.2: MULTOPS

is done by dropping packets from the sources.

In victim-oriented mode, the method will determine if an IP address or subnet is under attack by looking at the ratio of incoming vs outgoing packets. If this ratio becomes greater than a given threshold (i.e *incoming packets* >> *outgoing packets*), dropping packets against this destination address will mitigate the attack. However, this method will also cause «collateral damage», which means both malicious packets and legitimate packets will be dropped. This technique is also called *black-holing* or *null-routing* a destination.

In attacker-oriented mode, the method will determine the sources of an attack by looking at the ratio of received and sent packets from a source's point of view. If the ratio of received packets vs sent packets is below a given threshold (i.e *sent packets* >> *received packets*), dropping these packets might mitigate the attack. This method, on the other hand, is vulnerable to IP spoofing.

2.3 Source IP Monitoring

In [11], Peng et al. proposes a method for detecting distributed denial of service attacks based on findings in [6] by Jung et al. This method aims at detecting attacks by monitoring the increase of new IP addresses. The method contains two parts; an off-line training part, where IP addresses are added to a database, as they are shown to be legitimate. In order to decide if an address is legitimate or not, a simple rule such as connections with less than 3 packets is considered anomalous. The second part is the detection engine, which is based on monitoring the number of IP addresses within a time interval. This number is compared with the database, and then the number of new hosts is calculated. If the number of new hosts exceeds a given threshold, an alarm is set to indicate a possible attack. If an attack is not detected, the new IP addresses are added to the database, assuming they are shown to be legitimate. In the case of an attack, no IP addresses are added to the database.

2.4 An introduction to learning automata

An automaton[9] is a mechanism that simply makes a decision. Now, let us assume there is someone who gives feedback and tells the automaton if its decision was good or bad, then the automaton has the possibility to learn from its actions.

Learning can be defined as change in behaviour through experience. So, by learning one can improve the behaviour over time.

A learning automaton is a decision making device which is able to operate in an unknown– and non-deterministic environment. It is able to learn by interacting with the environment and updating its strategy for choosing the next action based on the response from the environment. The automa-

ton chooses from a set of actions, and the response from the environment is *positive* or *negative*. Figure 2.3 illustrates the interaction between a two-action automaton and its environment. The automaton updates its probability for choosing its next action based on the response from the environment.

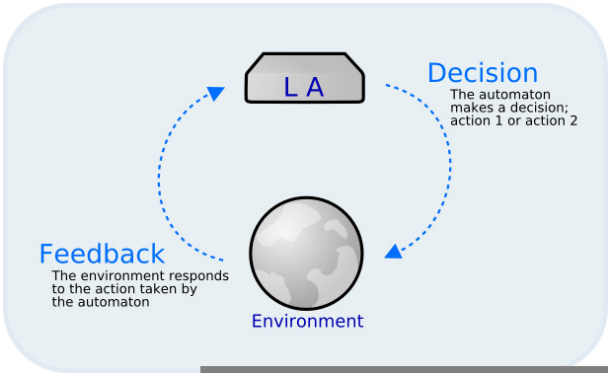

Figure 2.3: A learning automaton interacting with the environment

2.4.1 Learning algorithms

There are several algorithms for how to learn. The most researched classification is the *reinforcement algorithms*. These algorithms are based on the automaton being under constant supervision, ie. the automaton will always get a response from the environment for every action. However, it is assumed that the responses are not always correct, ie «the teacher» is not always correct. In real life, this situation is not uncommon. The teacher does not always give the student a correct answer, and the answer does not necessarily have to be correctly interpreted by the student. The following paragraphs cover the mainly used reinforcement algorithms.

Linear Reward Inaction

The linear reward inaction algorithm, or the L_{RI} algorithm, updates the probabilities only when it receives positive response from the environment. In the case of negative response from the environment, penalty will be ignored.

This algorithm is said to be expedient and ϵ-optimal in stationary random environments, which means that it performs better than choosing purely random actions. However, because of its non-ergodic behaviour, it is possible to get stuck in an absorbing state. This typically occurs when a probability tend to one. If the behaviour pattern should change at this point, it would take a long time to adapt the probability values to the environment. This algorithm is also sensitive to its initial probability values.

Linear Reward Penalty

The linear reward penalty algorithm, or the L_{RP} algorithm, updates the probabilities for both positive and negative responses. This algorithm has also shown to be expedient, but it is also ergodic. This means that it will not get stuck in an absorbing state, and it is not sensitive to its initial probability values. This behaviour can be an advantage in non-stationary environment, as it quickly adapts when the environment changes.

Linear Reward ϵ-Penalty

This algorithm is similar to the L_{RP} algorithm, except that the penalty learning rate is lower than the reward learning rate. Because of this, the behaviour changes to ϵ-optimal as well as ergodic, which means it will perform as optimal as it can.

Chapter 3

Network traffic collector

The network traffic collector is the first part of the proposed solution. Its objective is to perform high speed network traffic accounting and create traffic profiles for later analysis.

3.1 Requirements

The following requirements were set for the network traffic collector:

- Must be able to run on the i386 Linux platform.
  As Telenor[15] provided me with this hardware, the prototype will be developed on this platform.
- Must be able to do real-time packet capturing.
  The packet capturing must be done in real-time by listening on network interface card(s).
- Must be able to handle gigabit/s packet rate without significant drops.
  DDoS attacks are performed by sending packets in very high speed. In this thesis, the hardware limitations are set to 1-2Gbit/s.
- Must be able to decode packets up to transport layer (layer 4).
  For analysis purposes, the proposed method should perform detailed accounting up to transport layer, which include TCP, UDP, ICMP, etc.
- Must be able to export captured data.
  The data has to be exported as a traffic profile for later analysis by the *traffic profile analyser*.

3.2 Delimitations

The network traffic collector will only handle IPv4 packets. Packets using another network layer protocol will be ignored by the collector.

3.3 Creating traffic profiles

A traffic profile attempts to describe the traffic pattern at a specific location, in a specific time slot. The goal of the network traffic collector is to create these profiles.

In order to describe traffic, a set of attributes must be selected. For the traffic profile proposed in this thesis, the following attributes will be included:

- Packet and byte count for sent- and received IP packets.
- Packet and byte count for sent- and received TCP packets.
- Packet and byte count for sent- and received UDP packets.
- Packet and byte count for sent- and received ICMP packets.
- Packet and byte count for sent- and received for OTHER layer 4 protocols.

These statistics will be collected during a 1 minute time window. At the end of each traffic profile, a new 1-minute-profile will be created, and the recently populated profile will be exported to a database for analysis. This process is described in algorithm 2.

Algorithm 2 Creating traffic profiles

```
Require: Profile currentProfile
Require: Profile previousProfile
  loop
    currentProfile ← init new profile
    repeat
      populate currentProfile
    until time interval is reached
    previousProfile ← currentProfile
    export previousProfile {non-blocking}
  end loop
```

3.4 Sampling algorithm

DDoS attacks are usually aimed at a specific victim. Therefore, it is likely that under an attack, most of the traffic against the home network is directed against the victim. The optimal sampling solution in the case of detecting the attack would be sampling all packets against the victim. Since it is very difficult to know this prior to an attack, another solution could be sampling all packets on the network. The downside of this method is that it would exhaust the resources of the monitoring device on a busy network.

The *sample-and-hold* algorithm described in section 2.2.1 on page 7, aims at identifying large flows and ignoring the small and is well suited for this kind of application. I will use this sampling algorithm in order to identify the «popular destination» in the home network.

3.5 Data-structure

The data-structure used in the collector is based on the MULTOPS data-structure, which is described in detail in section 2.2.2 on page 9. One of the biggest advantages of the MULTOPS data-structure is the $O(4)$ lookup performance for a single IP address. In order to keep the number of «dropped» packets to a minimum, an efficient lookup algorithm is therefore important when performing high speed traffic accounting.

Another advantage of the MULTOPS data-structure is that it keep statistics for subnet prefixes at different aggregation levels. This allows me to manage the IPv4 address space using CIDR notation, which is more efficient than handling significally larger sets of single IP addresses when considering reactive methods such as IP filtering on routers. The more filters you apply on your router, the more it affects the router's performance. If you wish to deny traffic from `192.168.0.[0..255]`, clearly it makes more sense to block the `192.168.0.0/24` subnet rather than blocking single IP addresses `192.168.0.0`, `192.168.0.1`, `192.168.0.2`, ..., `192.168.0.255`.

As previously mentioned, the original MULTOPS heuristic monitored real-time packet rate statistics. For this method I have chosen to collect more detailed statistics, such as the protocol distribution. Therefore, as shown in figure 3.1 on the following page, the following extensions to the data-structure has been made:

- The `Record` will include the following statistics:
  - Received `IP` packet count
  - Sent `IP` packet count
  - Received `IP` byte count
  - Sent `IP` byte count
  - ... (same for `TCP`, `UDP`, `ICMP` and `OTHER`.

A complete list of the `Record` structure attributes can be found in appendix A on page 46.

Also, unlike the original MULTOPS heuristic, which continuously monitor incoming- and outgoing packet rates, the algorithm proposed in this thesis will collect detailed traffic statistics for a given time interval and export this as a traffic profile. At the end of each interval, the traffic profile will be exported to a database, while a new profile is created and initialised for further accounting.

As mentioned in section 2.2.2 on page 9, the original MULTOPS method has two «operating modes»; victim-oriented mode or attacker-oriented mode.

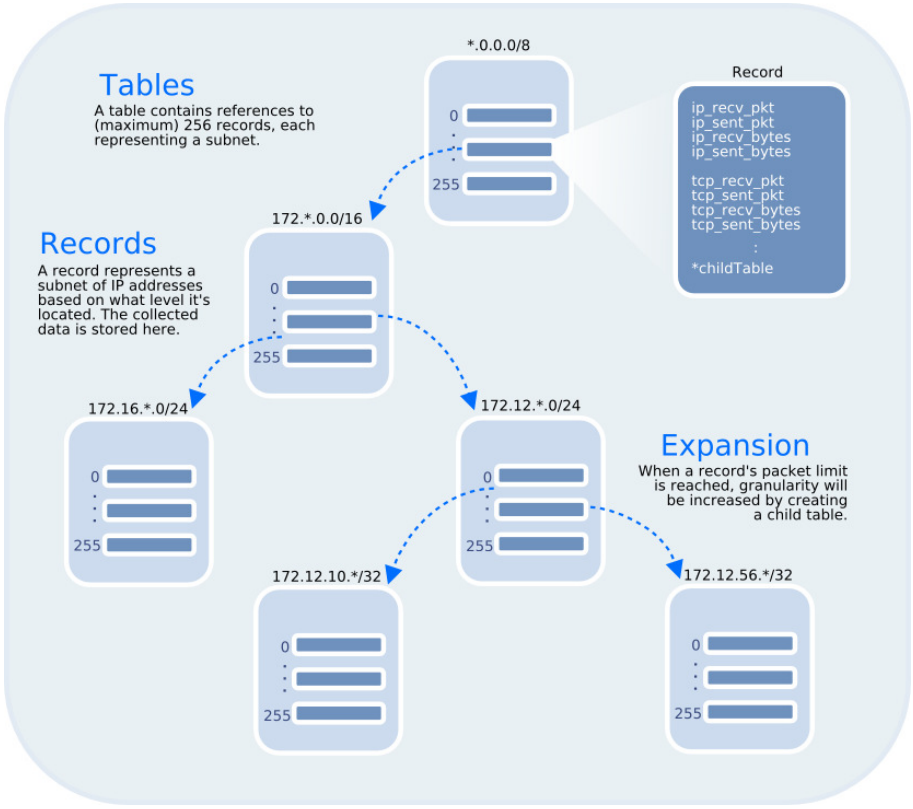

Figure 3.1: Extending MULTOPS

Both of these modes has their strengths and weaknesses, which is described in section 2.2.2 on page 9. The method proposed in this thesis attempts to take advantage of both of these modes by splitting the accounting process in two phases.

The first phase operates in victim-oriented mode by performing detailed accounting on IP addresses defined by the home network. When a packet destined for an IP address located in the home network, record attributes such as:

- `ip_recv_pkt`
- `ip_recv_bytes`
- `tcp_recv_pkt`
- `tcp_recv_bytes`
- ...

are incremented. And for outgoing packet originating from the home network, the following record attributes are updated:

- `ip_sent_pkt`
- `ip_sent_bytes`
- `tcp_sent_pkt`
- `tcp_sent_bytes`
- ...

The second phase of the accounting algorithm is attacker oriented. However, I have chosen to call this mode *«selective» attacker mode* as it is not equal to MULTOPS attacker mode. The difference between the two modes is that *selective* attacker mode only does accounting for source addresses that are connected to destinations accounted in victim mode. This is due to the chosen sampling algorithm described in section 3.4 on page 15. Source addresses of packets not directed against a sampled destination will therefore not be accounted. In other words, only source addresses that send packets to a top destination will be sampled. The `Record` updating algorithm can be described with the pseudo code shown in algorithm 3 on the next page, and algorithm 4 on the following page shows how the two accounting phases are combined. Figure 3.2 on page 20 gives an illustration of how the algorithms work together to build a traffic profile.

The tree behaviour is similar to the behaviour described in 2.2.2 on page 9. There is one significant difference however, since I am not tracking packet rate statistics in a continuous manner, but rather sampling packet statistics (incrementing values) in discretised time slots, the tree will never contract at any time, only expand within a profile's time slot. The tree will expand when a packet count or packet byte count exceeds a threshold value for the given traffic profile. The expansion algorithm can be described with the pseudo code given in algorithm 5 on the next page.

Algorithm 3 Record updating algorithm

```
Require: Record rec
Require: Packet pkt
  if incoming then
    increment rec's ip_recv_pkt
    add pkt.size to rec's ip_recv_bytes
    increment rec's tcp_recv_pkt
    ⋮
  else {outgoing}
    increment rec's ip_sent_pkt
    add pkt.size to rec's ip_sent_bytes
    increment rec's tcp_sent_pkt
    ⋮
  end if
```

Algorithm 4 Accounting phases

```
Require: Record recDestination
Require: IP destination
  if sample all packets to destination then
    recDestination ← get record for destination
    update recDestination {phase 1}
    update recDestination's MULTOPS source tree {phase 2}
  end if
```

Algorithm 5 Expansion algorithm

```
Require: Record rec
Require: Table recParent
  if rec's ip_recv_bytes > limit then
    if rec has no child table and recParent is not deepest level then
      create new child table under rec
    end if
  end if
```

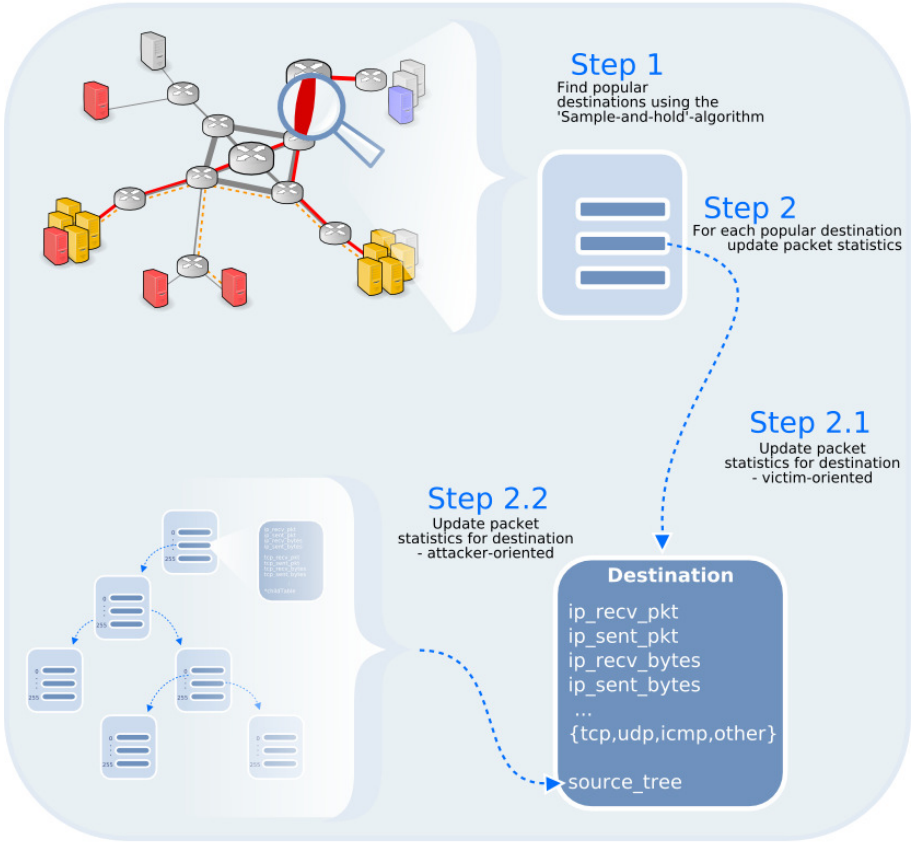

Figure 3.2: Building a traffic profile

3.6 Exporting traffic profiles

As the traffic profiles are created by the network traffic collector, they need to be exported to a database for further analysis. The traffic profile database is very simple and is built up by 5 tables; `src_acc` which holds «attacker oriented» accounting statistics, `dst_acc` containing «victim oriented» statistics, a `profile` table holding start- and end time-stamps for each profile, and `src` and `dst` tables holding source IP addresses with ip prefix and destination addresses, respectively. Their relations are illustrated in figure 3.3.

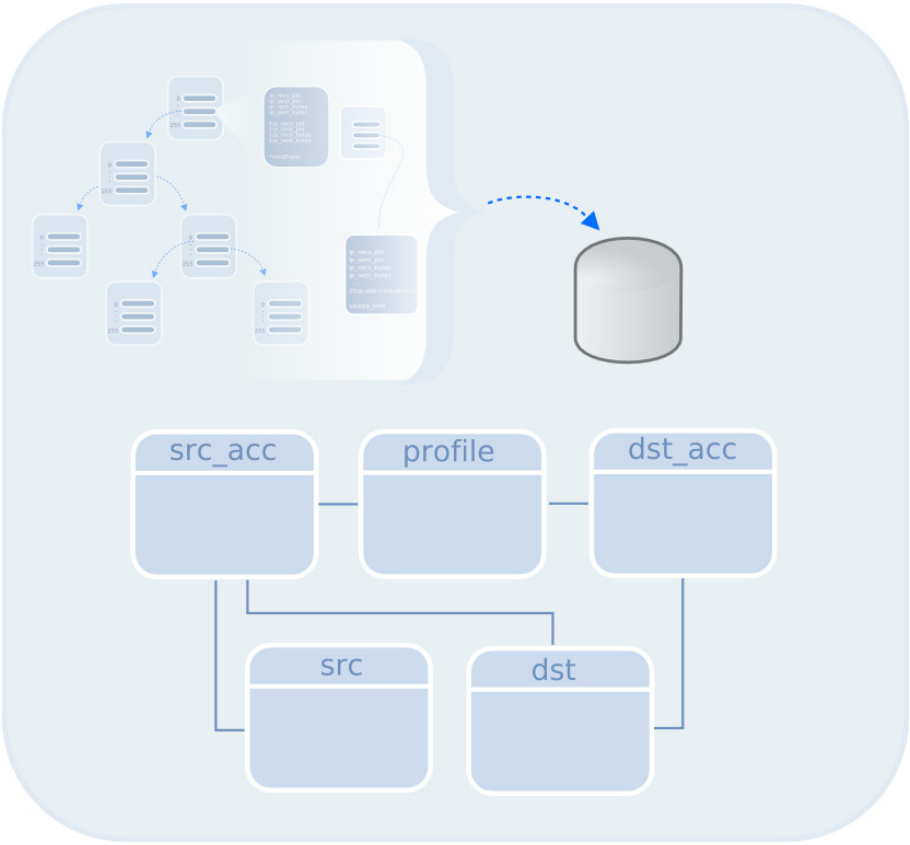

Figure 3.3: Exporting traffic profiles

Chapter 4

Traffic profile analyser

The second part of this thesis consists of developing a traffic profile analyser. Its task is to iterate through all profiles generated by the network traffic collector and look for anomalies and indications of attack traffic.

4.1 Attack characteristics

In order to detect an attack, some characteristics of hostile traffic is needed. Based on personal experience as a security analyst at Telenor Security Operation Centre[15] (TSOC), DDoS tools found on compromised hosts and the increased popularity in recursive DNS attacks[17], large UDP packets against a victim with very few, or none, packets in opposite direction can be considered as attack traffic with high probability. This is of course not necessary valid for all traffic on the Internet, so a few false positives should be taken into account.

Considering the TCP protocol, connections with less than 3 packets can also be flagged as possible attack traffic. This is also one of the rules used by Peng et al. in [11].

Another characteristic of attack traffic, as observed near the victim, is a sudden rise in incoming packets. This can be detected by monitoring the changes in packet- and byte count for each popular destination. However, this method is prone to false positives in the case of a *flash crowd*. A flash crowd event occurs when a large amount of hosts are visiting the same web site simultaneously, and will therefore have the same characteristics as an attack, considering incoming packet- and byte count. In an attempt to distinguish flash crowds from actual attacks, we make an assumption that normal traffic packet rate in one direction is proportional to the packet rate in the opposite direction. If a dis-proportionality occurs, this can be considered as a possible attack.

In [6], Jung et al. has observed that in the case of a flash crowd, most of the source addresses are already known, ie. they have visited the web site previously, while in the case of a DDoS attack, most of the source addresses are new to the web site. This is also a characteristic which will be taken into account in the proposed method.

4.2 Source IP Monitoring

In [11], Peng et al. proposed a method of detecting DDoS attacks using source IP address monitoring. This method is based on the findings in [6], and monitors the increase in new addresses over a certain period. The method is described in detail in section 2.3 on page 11 and has some similarities with the method proposed in this thesis. We both monitor source IP addresses and, at some point, decides whether they are participating in an attack or not. However, an important difference between the two is that the method proposed in [11] only monitor *single* IP addresses, while in this method, due to the MULTOPS datastructure, source IP addresses in addition to their subnets are monitored.

Because of the mentioned difference between the method proposed in [11] and this method, another learning- and decision method must be used regarding the source IP address monitoring. Consider the following example: A host at `192.168.3.25` is visiting a web site at the protected network. The method proposed in [11] sees this as a new source, recognises this as legitimate traffic and is therefore added to the database of known source IP addresses. Now, if we try to combine this with the traffic profile proposed in this thesis, `192.168.3.0/24`, `192.168.0.0/16` and `192.0.0.0/8`, depending on expansion threshold values, will be added to the list of known source IP addresses. Next, assume that an attack is initiated from multiple sources in the following range; `192.127.0.0/16`. These hosts will not be recognised as attackers since `192.0.0.0/8` is already in the database of known source IP addresses.

In an attempt to solve this issue, I have chosen to let learning automata help to decide whether a source- IP address or subnet is legitimate or not. This method will be described in the following section.

4.3 Monitoring source IP addresses and subnets

To summarise, the following characteristics are considered when looking for attack traffic:

- Sudden increase in incoming packets
- Incoming/outgoing packet rate dis-proportionality
- Very short/incomplete TCP connections
- Increase in new source IP addresses

As mentioned in section 2.4 on page 11, learning automata are decision making devices that are able to operate under an unknown- and non-deterministic environment – which is an environment that matches the Internet. The purpose of the automaton to be used in this thesis is to decide whether a source IP address or subnet is participating in an attack or not.

For an automaton to be able to learn, it needs a teacher. This teacher will be an analyser that operates similar to the decision engine proposed by Peng et al. in [11]. The analyser will iterate through all records in the traffic profile and will consider disproportional packet rates and suspiciously short TCP connections as a sign of possible attack traffic. In addition, the analyser will consider large amount of ICMP packets and various rare protocols (`OTHER`) as anomalous.

The analyser, or the teacher, should recognise typical attack traffic patterns based on the accounting information given in the traffic profiles. This includes packet statistics on `IP`, `TCP`, `UDP`, `ICMP` and `OTHER`.

I assign one learning automaton to each source IP address or subnet. This automaton will represent the *threat value* for the given source IP address or subnet. Theoretically, I could assign four automata to each source IP address or subnet, each representing the threat value for `TCP`, `UDP`, `ICMP` and `OTHER` protocol for the given source, respectively. The advantages of «splitting» the automaton into four parts is to differentiate between, for example, legitimate `TCP` traffic and hostile `UDP` traffic. However, by using a single automaton per source, all protocols above `IP` will «share» threat value, which will simplify the analysis later on. This will be discussed further in section 7.3.4 on page 38.

The automaton work with the analyser as follows: When a new traffic profile has been created, each automaton will, based on its internal state (ie. threat value), decide whether its assigned source address is «good» or «bad». Next, the analyser looks at the actual traffic profile statistics, considering `IP`, `TCP`, `UDP`, `ICMP` and `OTHER`, and either agrees or disagrees with the automaton on its decision. If the automaton and the analyser consent, the analyser will give positive feedback to the automaton for job well done. If they, on the other hand, do not consent, the analyser will give negative feedback.

Based on whether the automaton receives positive or negative feedback from the analyser, this will affect the next decision it makes. How the automaton handles the feedback depends on its updating algorithm. In section 2.4.1 on page 12, I have given an overview of the mainly used reinforcement algorithms that are of interest. In this thesis I am using the L_{RI} algorithm, which means that on positive feedback, the automaton is rewarded, but will ignore negative feedback. The L_{RI} updating algorithm used in this thesis can be described with the pseudo code given in algorithm 6 on the next page. The reason for choosing this algorithm will be discussed in the following paragraph.

As described in section 2.4.1 on page 12, the L_{RI} algorithm has an absorbing behaviour, which is generally not suitable for dynamic environments. However, in this thesis I make the assumption that a source does not (usually) change its behaviour from «good» to «bad», or vice versa. As this algorithm has an absorbing behaviour, it is also sensitive to its starting conditions. This fits very well with the proposed method. The starting condition for the automaton is the «threat value», which is a value between 0 and 1. Threat values close to 0 indicates low probability of «bad» source,

Algorithm 6 Linear reward-inaction updating algorithm

Require: Decision α
Require: LearningRate a
Require: State p {probability of choosing bad traffic}
Require: Feedback β
  if β is positive **and** α is bad traffic **then**
    $p = p + a * (1.0 - p)$
  else if β is positive **and** α is good traffic **then**
    $p = p - a * p$
  else {β is negative}
    ignore negative feedback
  end if

while values close to 1 indicates high probability of «bad» source. I have chosen a starting value of 0.5, which means that all new source IP addresses and subnets will get a threat value of 0.5 once they are detected.

Once a new source is detected, the learning automaton will decide whether it is a «bad» or a «good» source. With a starting condition of 0.5, it will guess «bad» with a probability of 50%. Depending on whether the analyser consents with this decision, the automaton will be rewarded and updates its threat value, or it will simply ignore the feedback and stay at 0.5.

Next time the source appears in a traffic profile, the automaton will decide between «good» or «bad», based on its current threat value. If this value is below 0.5, it will choose «good» with a higher probability than choosing «bad».

Let us consider the example from section 4.2 on page 23 again where a host at `192.168.3.25` is visiting a web site. This host will now be detected as a new source, as well as its subnets `192.168.3.0/24`, `192.168.0.0/16` and `192.0.0.0/8`, and they all initially get a threat value of 0.5. As this source appears in several traffic profiles over time, its threat value will decrease, and the probability of choosing «good» increases. Next, an attack is initiated from the `192.127.0.0/16` network. The automaton at `192.0.0.0/8` will most likely decide «good» source because of its history, which is reflected by the threat value, but the analyser will not consent, and the automaton ignores the feedback. However, at some point, the automaton for `192.0.0.0/8` *will* choose «bad», and the analyser will consent, causing the automaton to be rewarded according to its updating algorithm. Note that the automaton will choose «good» with a higher probability than choosing «bad». At the same moment, `192.127.0.0/16`, `192.127.*.0/24` and `192.127.*.*` are detected as new sources (depending on the expansion threshold values), and will have an initial threat value of 0.5. Now, these new sources has a higher probability of choosing «bad» than `192.0.0.0/8`, which means that the new sources will increase their threat values above 0.5.

In order to block this attack, you want to create filtering rules. However, you want to maintain network performance, so you want to apply as

few filtering rules as possible, but still be able to block the attack. Let us assume there was 10,000 hosts participating in the attack behind the `192.127.0.0/16` network. Instead of creating 10,000 filtering rules, it is obvious that you rather create one rule for the entire `192.127.0.0/16` network, or maybe even the `192.0.0.0/8` network if the attack was coming from the `192.129.0.0/16` network as well. In the latter case, depending on how much legitimate traffic was coming from `192.0.0.0/8`, it could be reasonable to block the entire network. However, if there was much legitimate traffic from the `192.168.0.0/16` network, this would be reflected in the `192.0.0.0/8` threat value, indicating that blocking the entire `192.0.0.0/8` network would cause collateral damage.

As a final note, I will summarise some of the key points of this chapter. The idea behind using learning automata is to combine characteristics, such as packet rate dis-proportionality and occurrence of new sources, with the traffic profiles provided by the network traffic accounter. The point is that each automaton will be able to adapt itself to the traffic profiles and give an indication of whether a source is participating in an attack, or not. The unique properties of the learning automaton makes it able to handle «noise» (ie. when the analyser makes wrong decisions) quite well. As a consequence of the chosen L_{RI} algorithm, this also makes the method robust considering IP spoofing.

Chapter 5

Experimental setup and simulations

In this chapter I will describe the experimental setup and simulations conducted with the network traffic accounter and the traffic profile analyser. This includes my network setup and configuration in addition to simulating normal traffic and DDoS attack traffic.

5.1 Network setup

The following hardware was provided by Telenor[15]:

- 3 x IBM xSeries 1U servers equipped with Broadcom 1000SX network interface cards.
- 2 x Linksys gigabit switches using 1000SX.

Using this hardware I created a network environment as shown in figure 5.1 on the following page. I dedicated one server for traffic profiling, normal traffic generation and attack traffic generation, respectively.

The server dedicated to traffic profiling was equipped with two Broadcom 1000SX network interface cards, each receiving normal traffic and hostile traffic, respectively. In a production environment, it might not be recommended using a port mirroring solution as described in figure 5.1 on the next page due to a potential loss in network performance on the affected switch. A more appropriate solution would be using a passive network tap, such as the ones provided by [10], in order to eliminate this loss of performance. However, using a passive tap causes incoming packets to be received on one card, while the outgoing traffic would be received by a second card.

5.1.1 System configuration

Another advantage of assigning incoming- and outgoing packets to a dedicated network interface card is the «load balancing» of interrupts. Under

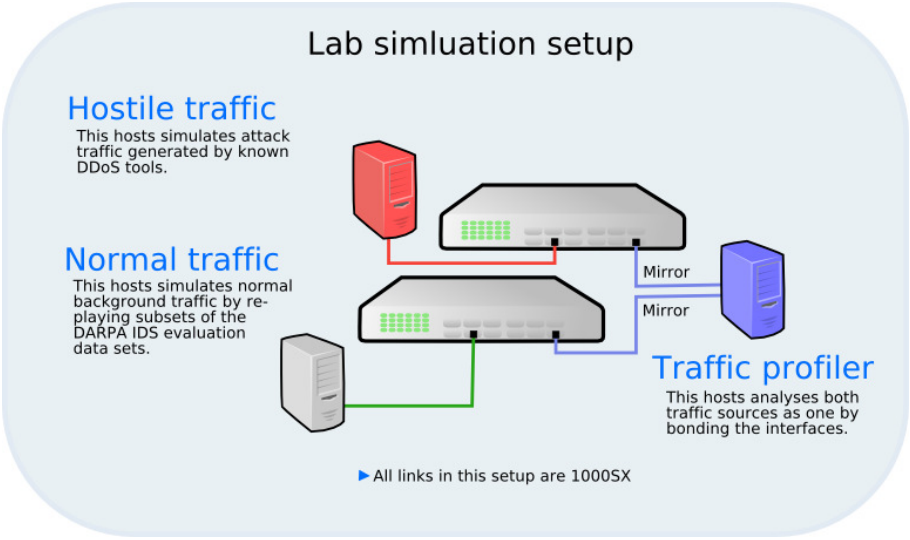

Figure 5.1: Network setup

an attack, the number of packets arriving at the card is massive, and this causes many interrupts to the CPU.

In this setup I had a dual Intel Xeon CPU with HyperThreading enabled, which allowed me to work against four logical CPUs in Linux. By overriding the kernel IRQ load balancing, I assigned a logical CPU to each network interface card. By doing this, the assigned CPUs would never be interrupted by anything else than network activity on their respective cards.

A third logical CPU was assigned to the traffic profiler application. Any other interrupt request was handled by the last logical CPU, which included the traffic profile analyser and general management.

5.2 Traffic simulations

Using the network setup as described in previous sections, simulations of network activity was needed. I will describe how this traffic was created in the following sections.

5.2.1 Simulating normal traffic

In order to simulate normal traffic, I used the DARPA IDS evaluation datasets[7] as a base. This traffic was injected into the gigabit link using `tcpreplay`[14], which is a tool designed to do exactly this.

In order to achieve high speed, I had to speed up the traffic, assuming that this would not affect the correctness of my experiment.

5.2.2 Generating high speed traffic

The Linux kernel offers a module called the Linux packet generator, or `pktgen`[16] for short. This is a tool for generating packets at a very high speed at the kernel level, which I used for testing the network traffic accounter performance.

However, the packets generated by this tool does not resemble «normal» attack traffic. To simulate a distributed attack, we want our source addresses to be random. Although this tool can be configured to use random source addresses, the traffic pattern it generates is too predictable because of its single threaded nature. The packet generator will generate a fixed number of packets with a random source address, then it will choose another source address and generate a fixed number of packets again. From the victim's point of view, it looks like the attacking sources are sending their packets in turn, and once an attacker is finished, it never sends a packet again. In real life, however, the sources are sending their packets *simultaneously*.

5.2.3 Simulating a distributed denial of service attack

To simulate a distributed denial of service attack, I first made a quick analysis of a DDoS attack tool which has been found on a compromised web server. This was a tool that generated small, non-spoofed UDP packets with random destination ports, depending on parameters given at startup time.

However, I have also experienced that UDP packets being part of an attack, are often much larger than packets generated by the tool mentioned above, eg. [17].

Therefore, in an attempt to make my simulated traffic blend in with normal traffic, I set the packet size to be approximately 600 bytes.

In order to make my traffic look distributed, I have to spoof the source address. Spoofing a packet's source address can be done with a tool called `hping`[13], which is a tool designed to create any arbitrary packet. Using this tool I created mid-size UDP packets with a spoofed source.

The remaining issue is to make `hping` continuously send packets with different source addresses. For this I wrote a small multi-threaded script which created a large array of random IP addresses. For each IP address in this array, I started a new thread with `hping`, generating UDP packets against my victim host with random destination port and with the given IP address as source.

5.3 Experiments

In this section I will describe the conducted simulations.

1. Background traffic replay
   In this experiment, I will measure the replay packet rate of the DARPA IDS evaluation datasets using two computers simultaneously.

2. Packet generation using `pktgen`
   In this experiment, I will measure the network traffic accounter performance using the `pktgen` tool from two computers simultaneously.

3. Combining background and attack traffic
   In this experiment, I will measure the network traffic accounter performance using two computers, each generating attack- and replaying background traffic, respectively.

4. Distributed denial of service attack
   In the last experiment, I will test the traffic profile analyser by replaying background traffic during the entire experiment from one computer. After 30 minutes, I will initiate a distributed denial of service attack from the second computer. The attack will last for 15 minutes.

Chapter 6

Results

In this chapter I will present the results of the conducted simulations described in the previous chapter.

6.1 Network traffic accounter performance

The network traffic accounter was confirmed to process background traffic (1) at a rate of ~600 Mbit/s, with an average packet size of ~520 bytes. This was the maximum packet rate for the background traffic I was able to replay. On the other hand, I achieved a much higher packet rate for the attack traffic. Using the `pktgen` module from the Linux 2.6 kernel (2), I was able to achieve a constant bit-rate of 1.24Gbit/s, with spikes up to 1.8Gbit/s. Combining the background traffic and the `pktgen` generated traffic (3), I was able to generate/replay packets at a rate of ~1.0GBit/s. Combining background traffic and the multi-threaded `hping` DDoS simulation, I was able to generate/replay packets at a rate of ~700MBit/s. All of the cases above were processed by the network traffic accounter without significant packet drops.

6.2 Traffic profile analyser

When it comes to general attack detection, the attack simulation conducted in this thesis was easily spotted by a simple bandwidth graph for each popular host. As shown in figure 6.1 on the following page, which is taken from a simple web-based bandwidth monitor interface developed during this work, we see a sudden increase in bandwidth for the victim at `172.16.114.50`. We also see that other hosts in the same network was experiencing a significant packet drop. The host on top in figure 6.1 is being attacked, while other hosts (shown below) are experiencing significant packet drops.

During the traffic profile analyser test (4), all actions made by the learning automata were recorded for analysis. In this experiment, ~1400 different sources were detected (this includes `/8`, `/16`, `/24` and `/32` IP prefixes). For simplicity, I will focus on two IP addresses in the same `/8` subnet, one

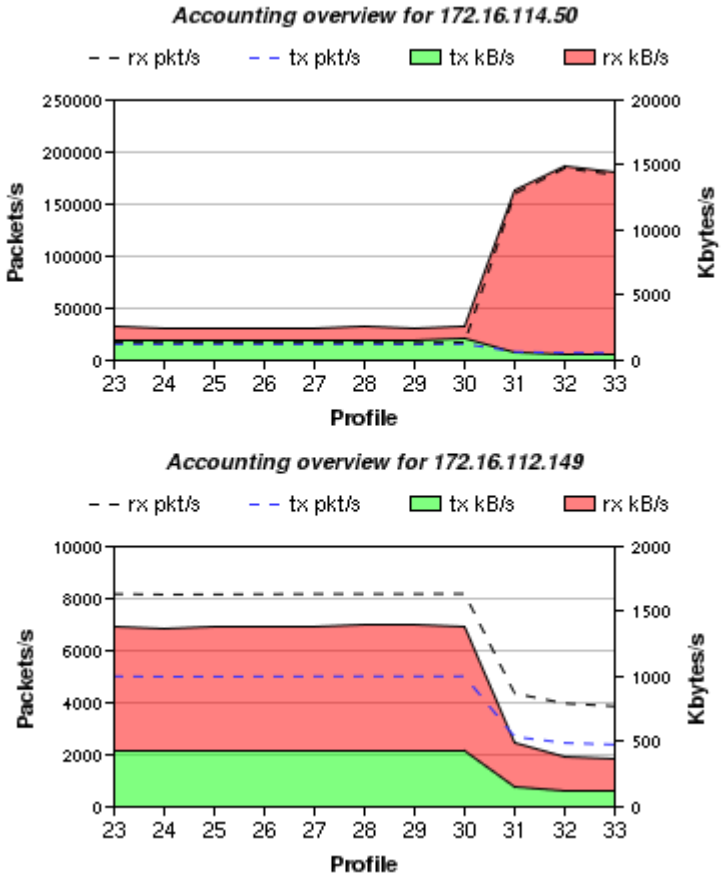

Figure 6.1: Accounting overview during an attack.

generating normal traffic, while the other one is participating in the attack; `161.181.250.169` and `161.80.155.233`, respectively. Figure 6.2 illustrates how the two IP addresses share the same /8 subnet and its learning automaton.

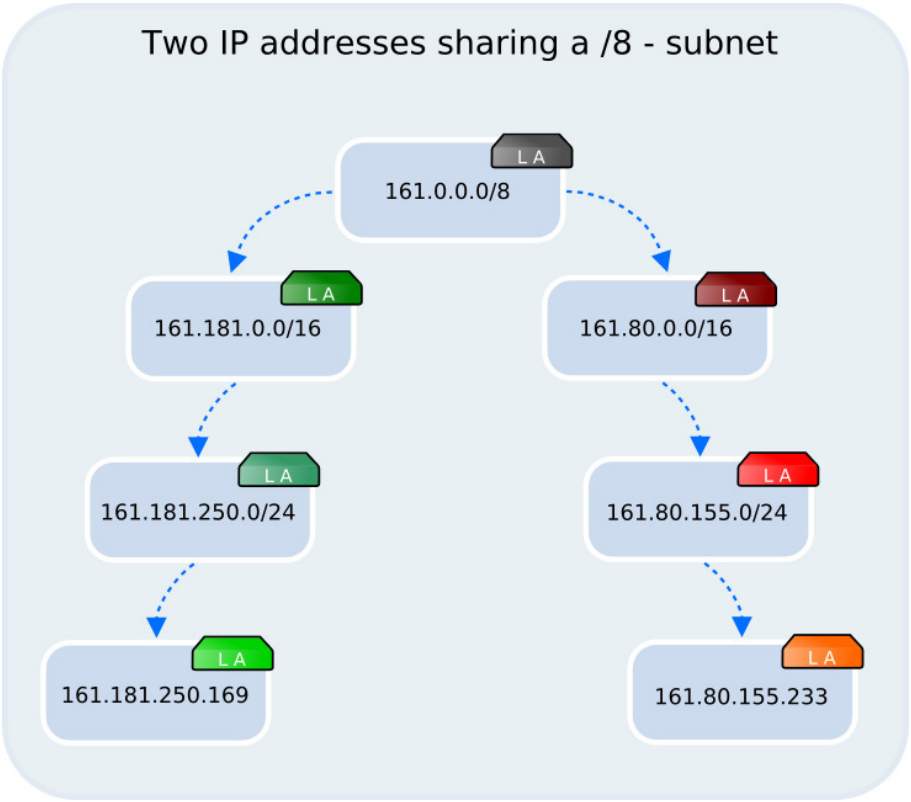

Figure 6.2: Two IP addresses sharing a /8-subnet

In this experiment, the threshold value for expanding the data-structure was set to 1 packet. This means that when a packet from `161.181.250.169` arrived, `161.0.0.0/8` was accounted, next `161.0.0.0/8` and `161.181.0.0/16`, and so on. Because of this, we see all subnets of the IP address already at the first profile (since it sent 4 or more packets). Each subnet, as described in section 4.3 on page 23, has its own learning automaton.

Figure 6.3 on the following page shows how the two IP addresses' and subnets' learning automata developed over time. In the first 30 profiles, we only see the subnets connected to the `161.181.250.169` address, as expected. We also see that this traffic is recognised as *normal* traffic, and that the automata were rewarded when they made the correct decision (according to the analyser). Sometimes, the automata made the wrong decisions, in these cases penalties were ignored and the threat values remained unchanged.

Since there is nothing else affecting the learning automata for `161.181.250.169` and its subnets, they will eventually converge to the same threat value.

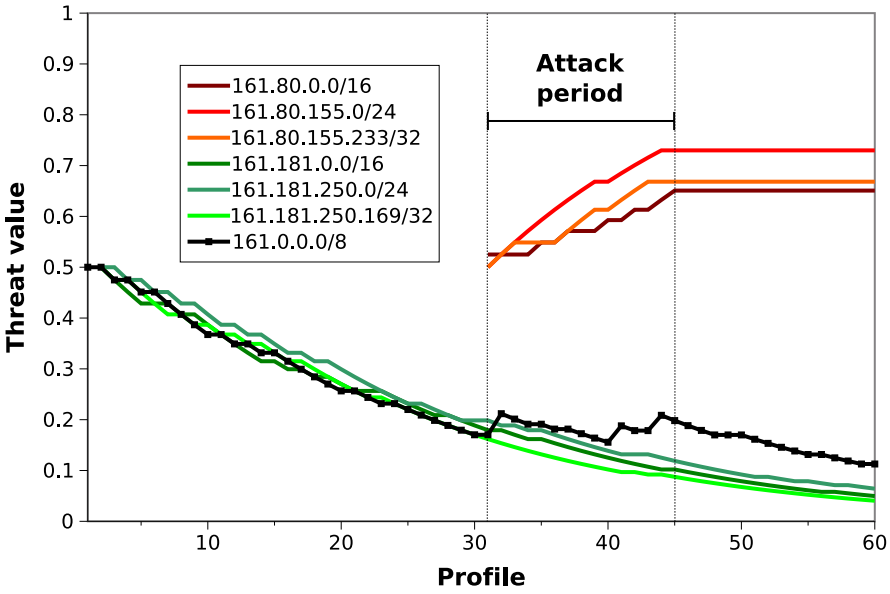

Figure 6.3: Profile development

After the 30^{th} profile, `161.80.155.233` is added to the source list. As expected with new sources, the threat value was initialised to 0.5. And, as with `161.181.250.169`, all subnets were added within the same profile, however the `161.0.0.0/8` is already existing.

As expected, `161.80.0.0/16`, `161.80.155.0/24` and `161.80.155.233/32` increased their threat values as their traffic was recognised as hostile. On the other hand, `161.0.0.0/8` was now affected by both attack traffic and normal traffic. During the attack period, we see that `161.0.0.0/8` is increasing slightly. However, due to its history it will not increase in the same speed as `161.80.*.*/*`. It is also considered a bad candidate for blocking due to the low threat value.

Once the attack period is over, we see that `161.0.0.0/8` is instantly decreasing its threat value. `161.181.0.0/16`, `161.181.250.0/24` and `161.181.250.169/32` has during the attack period been recognised as normal traffic, which is reflected by their decreasing threat values.

Chapter 7

Discussion

In this chapter I will discuss the work that has been done in this thesis, and the results that were achieved during the simulations.

7.1 Traffic simulations and characteristics

The background traffic simulations conducted in this thesis was based on the DARPA IDS evaluation datasets from 1999. Based on the signatures/rules of the traffic profile analyser, none of the background traffic was detected as possible attack traffic. However, in a production environment which has an up to date traffic scenario, including «new» services such as Voice over IP, P2P and streaming traffic, one should expect false positives (ie. noise) due to some dis-proportionality in the protocols. E.g. when streaming audio/video from a server, most of the packets are incoming, from a client's point of view. This can probably be handled by creating rules for these traffic patterns, but it requires a more in-depth study on those protocols than what has been conducted in this thesis.

The attack traffic simulations in this thesis was mainly based on experiences from TSOC and DDoS tools found on compromised computers. Observations of recent DDoS attacks shows that UDP traffic is being used in combination with DNS reflection- and amplification attacks[17]. In DNS amplification attacks, the number of attacking hosts (actually legitimate third-party DNS servers) are usually lower than the number of compromised hosts in a botnet. However, when it comes to detecting this traffic, the same method can be applied when detecting any other sources participating in an attack. There is one catch, though; the «attacking» DNS servers could be used for legitimate purposes by the victim network. By blocking all traffic from these DNS servers, collateral damage might be inflicted. In order to mitigate such attacks, where both attack- and normal traffic is originating from the same host, a deeper packet inspection could be performed in order to distinguish between «good» and «bad» packets. Since deep packet inspection like this can not be performed by normal routers, an in-line device which is able to block packets based on deep packet analysis, is required. However, deep packet inspection can be very resource intensive,

and would not perform very well at high speed. In the case of DNS amplification attack, another option that would not require an in-line device could be blocking or rate-limiting unusually large UDP packets, coming from the respective DNS server. Although this option would likely mitigate attacks using very large DNS responses, it would also cause some collateral damage due to cases where legitimate DNS responses in fact are large. On the other hand, in the case where the DNS server is flooding the victim with normal sized DNS responses, collateral damage is usually inevitable without a deep packet inspecting in-line device.

7.2 Data-structure

The data-structure in this thesis is based on MULTOPS. The MULTOPS data-structure was designed to be a lightweight structure for «live» monitoring of traffic rates at different aggregation levels. The major difference with MULTOPS and the data-structure used in this thesis is that MULTOPS operates in a continuous manner with a single instance of the data-structure, while in this thesis a new data-structure is created at every 1 minute time slot. Another major difference between the two methods is the amount of information that is being collected. The method proposed in this thesis collects information up to the transport layer. By doing this, the traffic profile analyser has a much broader basis for correctly detecting attack traffic than simply monitoring incoming- and outgoing traffic rates. The reason for this is that the traffic profile analyser can have multiple signatures or rules, which describes attack traffic at a more detailed level than the only rule looking for disproportional traffic rates.

One of the advantages with MULTOPS is that it is able to operate within a fixed memory budget. To be able to do this, it contracts when it reaches its upper memory limit. The structure used in this thesis, on the other hand, does not have this functionality. For each structure that is created in a time slot, the structure will only expand, until the next time slot arrives. At the start of each time slot, a new, empty data-structure is created. Because of this, and the fact that it is collecting much more information, the structure used in this thesis is more resource intensive. However, I do not find this of big concern, since the traffic profile computer has a much higher memory budget than the network devices MULTOPS was originally designed for.

When it comes to response time, MULTOPS will report and react to anomalous behaviour once it is detected, while the proposed method in this thesis will not report until it has reached the end of the current time window. This creates a possible response delay of 1 minute when using 1 minute time slots. The time slots could be decreased for the costs of more resources, however I find that a 1 minute delay is acceptable.

7.3 Source behaviour monitoring and learning automata algorithms

In this section I will discuss the choice of learning automata algorithm used in this thesis, and how it would react to different types of attacks.

7.3.1 Learning automata algorithms

The learning automata algorithm used in this thesis is the L_{RI} algorithm. The main reason for choosing this algorithm was because of the assumption that the behaviour of source IP addresses would not change from «good» to «bad», or vice versa. This assumption was built on the work by Jung et al. in [6], which claims that under an attack, most of the sources are new to the victim.

The L_{RI} algorithm operates in a such way that it only updates its probability value for choosing an action when it receives positive feedback from the environment for the chosen action. The consequence of this is that if an automaton repeatedly chooses action «good traffic», and receives positive feedback from the environment (ie. the analyser recognises the traffic as normal) for this action, the probability of choosing action «bad traffic» moves towards 0. Therefore, in cases where the analyser recognises the traffic as «bad traffic», the automaton will most likely choose action «good traffic» (assuming the history stated above). However, it will receive negative feedback from the environment, causing the probability values to remain the same. What just happened is that the automaton interpreted the «bad traffic» as a mistake by the analyser, or noise, which might occur in a random environment. In order for the automaton to «change direction», the analyser must recognise traffic as «bad traffic» and the automaton must choose action «bad traffic» despite the lower probability.

An optional algorithm, like the L_{RP} algorithm, updates its probability values on both positive and negative feedback. A consequence of this is that the automaton much more quickly adapts to changes in behaviour than when using L_{RI}. This means that if the automaton's probability of choosing action «good traffic» is very close to 1.0 and the analyser recognises traffic as «bad traffic», the automaton would likely choose action «good traffic» and receive negative response from the environment. Although it receives negative response, the automaton will act on this penalty by increasing its probability of choosing «bad traffic» next time.

A third algorithm, such as the $L_{R\epsilon P}$, will have the same characteristics as the L_{RP}. However, in the case of a penalty the learning rate would be lower than in the case of a reward for the $L_{R\epsilon P}$. Because of this, automata acting on penalties can be more sensitive to noise and IP spoofing. I will discuss the problem of IP spoofing in the following section.

7.3.2 IP spoofing

Because of this absorbing behaviour of the chosen L_{RI} algorithm, the method will be fairly robust against IP spoofing attacks, where the goal is to disrupt normal traffic by attempting to influence the learning automata for the respective sources.

So, let us assume that the attacker has knowledge of how this method works and what sources are using the target's services. His goal is to disrupt the normal activity, so the attacker commands his bots to send arbitrary packets against the victim, using random source addresses that are *known* to the victim. This scenario would contradict the findings in [6] by Jung et al., however a highly targeted attack *could* be arranged like this. To the victim, it appears that visiting hosts has changed their behaviour, and the learning automata are having a hard time «changing direction» of the source behaviour.

So how do you mitigate a such highly targeted attack? This is very difficult. One option could be using another algorithm for our learning automata, such as the L_{RP} which adapts much more quickly to changes in behaviour. However, by doing to this, you might end up blocking everyone who usually visits your site, which in itself causes a denial of service. The proper way to handle this would be applying ingress filtering on all edge routers in your (or your upstream ISP) network, which would effectively drop spoofed packets before doing too much damage.

7.3.3 Sources changing behaviour

Another problem arises if the sources behaviour actually *does* change, and not because of a spoofing attack. Although this contradicts the findings in [6] by Jung et al., it does not mean it can not happen. An important difference between the two scenarios, targeted IP spoofing and changing source behaviour, is that the traffic coming from a IP spoofing attack can potentially generate a much higher traffic rate than sources which happen to be infected by bot agents in addition to be visitors of the victim site. In these cases, it might be sufficient to block the sources that are attacking the victim and being new to the victim, but allowing traffic from the known- but attacking hosts. If the remaining attack traffic rate is high enough to sustain the denial of service, another method than the one proposed in this thesis should be used, or try increasing the number of learning automata per source as described in the following section.

7.3.4 Increasing the number of learning automata

In cases where you might wish to block hosts generating both attack- and normal traffic, a possible option could be increasing the number of automata per source. By doing this, each source could have one automaton representing the threat value for `TCP`, one for `UDP`, one for `ICMP` and one for `OTHER`. This would also require changes to the traffic profile analyser,

as it would need to reward or penalise the respective automaton, based on what protocol the traffic is based on. Using several automata per source, you have the possibility to distinguish between e.g. normal TCP traffic and attacking UDP traffic from the same source.

7.3.5 Expiration of learning automata states

The method proposed in this thesis does not currently expire the states of the learning automata. This means that as the time goes by, each source will have an infinite past experience. Due to expiring DHCP leases, it might be sensible to expire (ie. reset), the learning automaton's probabilities if its respective source has not been observed within a given time period. By not expiring these states, there is a chance that a source will suddenly «change its behaviour» because someone else is currently using the IP address due to the fact that a DHCP lease expired.

Chapter 8

Conclusion and further work

8.1 Conclusion

In this thesis I have proposed and developed a method for detecting- and identifying the sources of a distributed denial of service attack. There has been much research in these areas and I have taken advantage of several previously proposed solutions in the areas of traffic measurement and detection- and identification of denial of service attacks. I have focused on their strengths and combined them into a robust and efficient method for creating- and analysing traffic profiles. The proposed solution consists of two parts; a network traffic collector and a traffic profile analyser.

My contribution to the area of DDoS detection and identification has been exploring the usage of learning automata for tracking source IP address- and subnet behaviour over time.

I have evaluated the proposed method in a simulated network environment using the DARPA IDS evaluation datasets as background traffic, while simulating a distributed denial of service attack based on existing DDoS attack tools.

The results of the simulations conducted in this thesis shows that the information provided by the traffic profiles, created by the network traffic collector, serves as a satisfying base for detecting attacks.

During the simulations, the traffic profiles also appeared to provide sufficient information for the analyser to separate «good» traffic from «bad» traffic, with a low error probability. However, it is not confirmed that the proposed method will work optimally in a production environment with various new protocols not included in the DARPA IDS evaluation datasets, without the need of modifying or adding rules for the analyser.

The results also shows that learning automata using the linear reward-inaction updating algorithm are able to make a clear indication of whether a source IP or subnet is participating in an attack or not, with satisfying accuracy. This algorithm has also shown to be robust against IP spoofing attacks aimed at influencing legitimate traffic, however the method has shown to be somewhat vulnerable to sources changing their behaviour from «good» to «bad». Although, this might be handled by expiring the learning automata

states after a while.

In the case of an attack, filtering rules can be created based on the current states of the learning automata. Since the proposed method is tracking source IP addresses and subnets, more efficient rules can be created based on subnets instead of multiple IP addresses.

8.2 Further work

In this section I will propose a few suggestions for further work.

- **Increase the number of automata per source**
  Currently, there has only been conducted simulations with a single automaton per source. It would be interesting to see how the method would perform when mixing attack- and normal traffic from the same source, using four automata per source.
- **Increase level of details in traffic profiles**
  Some attacks might be easier to detect and identify if other statistics were available to the analyser, such as port number and time-to-live distributions. It would be interesting to see how the network traffic accounter would perform when increasing the level of details.
- **Updated network environment**
  The DARPA IDS evaluations datasets from 1999 is outdated. It would be interesting to see how the proposed method would perform in a live production environment.

References

[1] Aditya Akella, Mukesh Agarwal, and Ashwin Bharambe. Distributed, profile-based ddos detection and mitigation in isp networks, 2004.

[2] Cisco. Cisco ios netflow - cisco systems. http://www.cisco.com/.

[3] C. Estan and G. Varghese. New directions in traffic measurement and accounting, 2001.

[4] S. Floyd, S. Bellovin, J. Ioannidis, K. Kompella, R. Mahajan, and V. Paxson. Pushback messages for controlling aggregates in the network.

[5] T. Gil and M. Poletto. Multops: a datastructure for bandwidth attack detection, 2001.

[6] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites, 2002.

[7] MIT Lincoln Laboratory. DARPA IDS Evaluation Data Sets. http://www.ll.mit.edu/IST/ideval/.

[8] J. Mirkovic, J. Martin, and P. Reiher. A taxonomy of ddos attacks and ddos defense mechanisms, 2001.

[9] Kumpati S. Narendra and Mandayam A. L. Thathachar. *Learning automata: an introduction*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[10] NetOptics. Netoptics: Network taps and aggregation solutions for passive network access. http://www.netoptics.com/.

[11] T. Peng, C. Leckie, and K. Ramamohanarao. Detecting distributed denial of service attacks using source ip address monitoring, 2002.

[12] Martin Roesch. Snort network intrusion detection system. http://www.snort.org.

[13] Sourceforge. hping security tool. http://www.hping.org.

[14] Sourceforge. tcpreplay tool, 2000. http://tcpreplay.sourceforge.net/.

[15] Telenor. Telenor Security Services. http://www.telenor.no/bedrift/sikkerhet.

[16] Linus Torvalds. Pktgen: Linux packet generator. http://linux-net.osdl.org/index.php/Pktgen.

[17] Randal Vaughn and Gadi Evron. Dns amplification attacks, 2006. http://www.isotf.org/news/DNS-Amplification-Attacks.pdf.

Glossary

Amplification attack A reflection attack where the attacker sends a spoofed «small» packet against a third-party service, where the response packet to the victim is larger than the «small» packet page 7

Bot Agent Compromised host controlled by the attacker. ..... page 1

Botnet A botnet is a large number of compromised computers that are used to create denial of service attacks or send spam page 1

Botnet Server A server which acts as a command and control center for a network of compromised computers. ..... page 1

Collateral damage In networking; unintendedly affecting legitimate network traffic ..... page 7

DDoS Distributed Denial of Service Attack ..... page 1

DHCP Dynamic Host Configuration Protocol ..... page 38

Flash crowd A flash crowd event occurs when a large amount of computer hosts are visiting the same web site simultaneously page 22

Inline device In terms of networking devices, an inline device, as opposed to a passive device, has the ability to block packets in addition to monitor. ..... page 35

IP Spoofing A technique used for indicating that a packet is coming from another host. This is engaged by modifying the source IP address field in the packet header ..... page 11

Learning automaton A decision making device which is able to operate in an unknown- and non-deterministic environment .. page 11

P2P Peer-to-peer is a networking type which allows computer users using the same P2P-protocol exchange files. Often used for downloading large files, such as movies and music ..... page 35

Reflection attack An attack method based on exploiting weaknsesses of the IP protocols using the IP spoofing technique .... page 7

Streaming A technology that enables playback of audio and video without downloading the entire file in advance ..... page 35

TSOC Telenor Security Operation Center ..... page 22

VoIP Technology used to transmit voice conversations over computer networks using the Internet Protocol ..... page 35

Zombie See *Bot Agent* ..... page 1

Appendix A

MULTOPS extended Record-structure

```
struct _Record
{
  /* ip packets stats */
  u_int32_t ip_recv_pkt;
  u_int32_t ip_sent_pkt;
  u_int32_t ip_recv_bytes;
  u_int32_t ip_sent_bytes;

  /* tcp packets stats */
  u_int32_t tcp_recv_pkt;
  u_int32_t tcp_sent_pkt;
  u_int32_t tcp_recv_bytes;
  u_int32_t tcp_sent_bytes;

  /* udp packets stats */
  u_int32_t udp_recv_pkt;
  u_int32_t udp_sent_pkt;
  u_int32_t udp_recv_bytes;
  u_int32_t udp_sent_bytes;

  /* icmp packets stats */
  u_int32_t icmp_recv_pkt;
  u_int32_t icmp_sent_pkt;
  u_int32_t icmp_recv_bytes;
  u_int32_t icmp_sent_bytes;

  /* other packets stats */
  u_int32_t other_recv_pkt;
  u_int32_t other_sent_pkt;
  u_int32_t other_recv_bytes;
  u_int32_t other_sent_bytes;
```

```
  /* pointer to child table */
  Table *child;
};
```